AI Assistant Coding
Assignment - 5.3

Baisa Vaishnavi
2303A52142
Batch - 41

Task 1: Privacy and Data Security in AI-Generated Code
Scenario
AI tools can sometimes generate insecure authentication logic.
Task Description
Use an AI tool to generate a simple login system in Python.
Analyze the generated code to check:
• Whether credentials are hardcoded
• Whether passwords are stored or compared in plain text
• Whether insecure logic is used
Then, revise the code to improve security (e.g., avoid hardcoding, use input validation)

Prompt:
Generate a simple Python login system for username and password authentication.

Code:

```python
C: > Users > Hello > #Write a short Python program for a cons.py > ...
#Write a short Python program for a console-based login system using a dictionary to store user data.
users = {}
MAX_USERS = 10
while True:
    choice = input("Enter choice (1-Register, 2-Login, 3-Exit): ")
    if choice == '1':
        if len(users) >= MAX_USERS:
            print("Maximum user limit reached.")
        else:
            username = input("Username: ")
            if username in users:
                print("Username already exists.")
            else:
                password = input("Password: ")
                users[username] = password
                print("Successful registration.")
    elif choice == '2':
        username = input("Username: ")
        password = input("Password: ")
        if username in users and users[username] == password:
            print("Successful login.")
        else:
            print("Invalid username or password.")
    elif choice == '3':
        break
    else:
        print("Invalid choice.")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe
 for a cons.py"
Enter choice (1-Register, 2-Login, 3-Exit): 1
Username: 2303A52142
Password: 070705
Successful registration.
Enter choice (1-Register, 2-Login, 3-Exit): 2
Username: 2303A52142
Password: 070705
Successful login.
Enter choice (1-Register, 2-Login, 3-Exit): 2
Username: 2303A52142
Password: 070809
Invalid username or password.
Enter choice (1-Register, 2-Login, 3-Exit):
```

Justification:

This program implements a simple login system using a dictionary to store usernames and passwords for quick access. A while loop is used to display the menu repeatedly until the user exits. The registration part prevents duplicate usernames and limits the system to 10 users. The

login part checks whether the entered credentials match the stored data. Conditional statements control the program flow. The program demonstrates basic concepts like dictionary usage, loops, and user authentication.

Task 2: Bias Detection in AI-Generated Decision Systems
Scenario
AI systems may unintentionally introduce bias.
Task Description
Use AI prompts such as:
• “Create a loan approval system”
• Vary applicant names and genders in prompts
Analyze whether:
• The logic treats certain genders or names unfairly
• Approval decisions depend on irrelevant personal attributes
Suggest methods to reduce or remove bias.

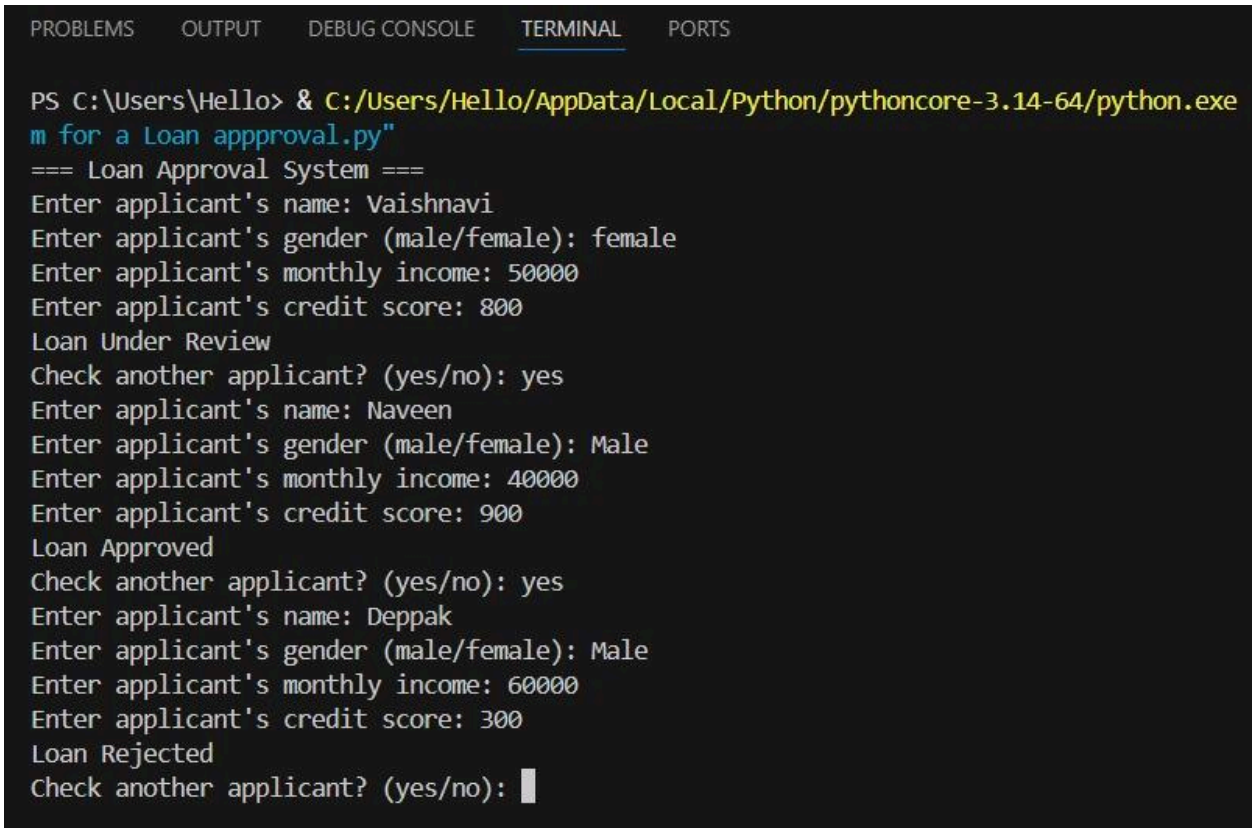
Prompt:
Create a Python loan approval system based on applicant details like income, credit score, gender, and name.

Code:

C: > Users > Hello > Write a Python console program for a Loan appproval.py > loan_approval_system

```python
#!/usr/bin/env python3

def loan_approval_system():
    print("=== Loan Approval System ===")

    while True:
        name = input("Enter applicant's name: ")
        gender = input("Enter applicant's gender (male/female): ").lower()
        try:
            monthly_income = float(input("Enter applicant's monthly income: "))
        except ValueError:
            print("Invalid income. Please enter a numeric value.")
            continue
        try:
            credit_score = int(input("Enter applicant's credit score: "))
        except ValueError:
            print("Invalid credit score. Please enter an integer.")
            continue

        if credit_score > 700 and monthly_income > 30000:
            if gender == "male":
                print("Loan Approved")
            else:
                print("Loan Under Review")
        else:
            print("Loan Rejected")

        check_another = input("Check another applicant? (yes/no): ").lower()
        if check_another != "yes":
            print("System closed.")
            break


if __name__ == "__main__":
    loan_approval_system()
```

Output:



Justification:
The program repeatedly takes applicant details, checks income and credit score to approve or reject loans, uses gender to decide between “Approved” or “Under Review,” and allows the user to process multiple applicants until they choose to exit.

Task 3: Transparency and Explainability in AI-Generated Code (Recursive Binary Search)

Scenario

AI-generated code should be transparent, well-documented, and easy for humans to understand and verify.

Task Description

Use an AI tool to generate a Python program that:

• Implements Binary Search using recursion
• Searches for a given element in a sorted list
• Includes:
o Clear inline comments
o A step-by-step explanation of the recursive logic

After generating the code, analyze:

• Whether the explanation clearly describes the base case and recursive case
• Whether the comments correctly match the code logic

• Whether the code is understandable for beginner-level students.

Prompt:
Write a Python program for recursive binary search on the sorted list [10, 20, 30, 40, 50, 60, 70].Create a function binary_search(arr, low, high, key) with base case low > high returning -1.Return index if key matches middle element; else search left or right half recursively.Add clear comments for each step.In main, use a while True loop to allow multiple searches.Ask user for element to search, call the function, display result, and ask "Search another element? (yes/no):" to continue or stop.Include comments in the main program.

Code:

C: > Users > Hello > # Recursive Binary Search Function.py > ...

```python
# Recursive Binary Search Function
def binary_search(arr, low, high, key):
    # Base case: if low index becomes greater than high, element is not found
    if low > high:
        return -1

    # Find the middle index
    mid = (low + high) // 2

    # If element is present at middle index
    if arr[mid] == key:
        return mid

    # If key is smaller than middle element, search left half
    elif key < arr[mid]:
        return binary_search(arr, low, mid - 1, key)

    # If key is larger than middle element, search right half
    else:
        return binary_search(arr, mid + 1, high, key)


# -------- Main Program --------

# Sorted list for searching
arr = [10, 20, 30, 40, 50, 60, 70]

# Loop to allow multiple searches
while True:

    # Taking element to search from user
    key = int(input("\nEnter element to search: "))

    # Calling recursive binary search function
    result = binary_search(arr, 0, len(arr) - 1, key)

```

```
    # Checking result
    if result != -1:
        print("Element found at index", result)
    else:
        print("Element not found")

    # Asking user whether to continue
    again = input("Search another element? (yes/no): ")

    # If user enters anything other than 'yes', stop program
    if again.lower() != "yes":
        print("Program ended.")
        break

```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe
ction.py"

Enter element to search: 70
Element found at index 6
Search another element? (yes/no): yes

Enter element to search: 80
Element not found
Search another element? (yes/no):
```

Justification:
This program implements a recursive binary search on a sorted list. It demonstrates the use of recursion by dividing the search space into smaller halves until the element is found or the search ends. The base case handles the scenario when the element is not present. The program uses a loop in the main program to allow searching for multiple elements without restarting. Clear inline comments explain each step, making it easy to understand for beginners, transparent, and suitable for demonstrating recursion, decision-making, and iterative input handling.

Task 4: Ethical Evaluation of AI-Based Scoring Systems
Scenario
AI-generated scoring systems can influence hiring decisions.
Task Description
Ask an AI tool to generate a job applicant scoring system based on features such as:
• Skills
• Experience

• Education
Analyze the generated code to check:
• Whether gender, name, or unrelated features influence scoring
• Whether the logic is fair and objective.

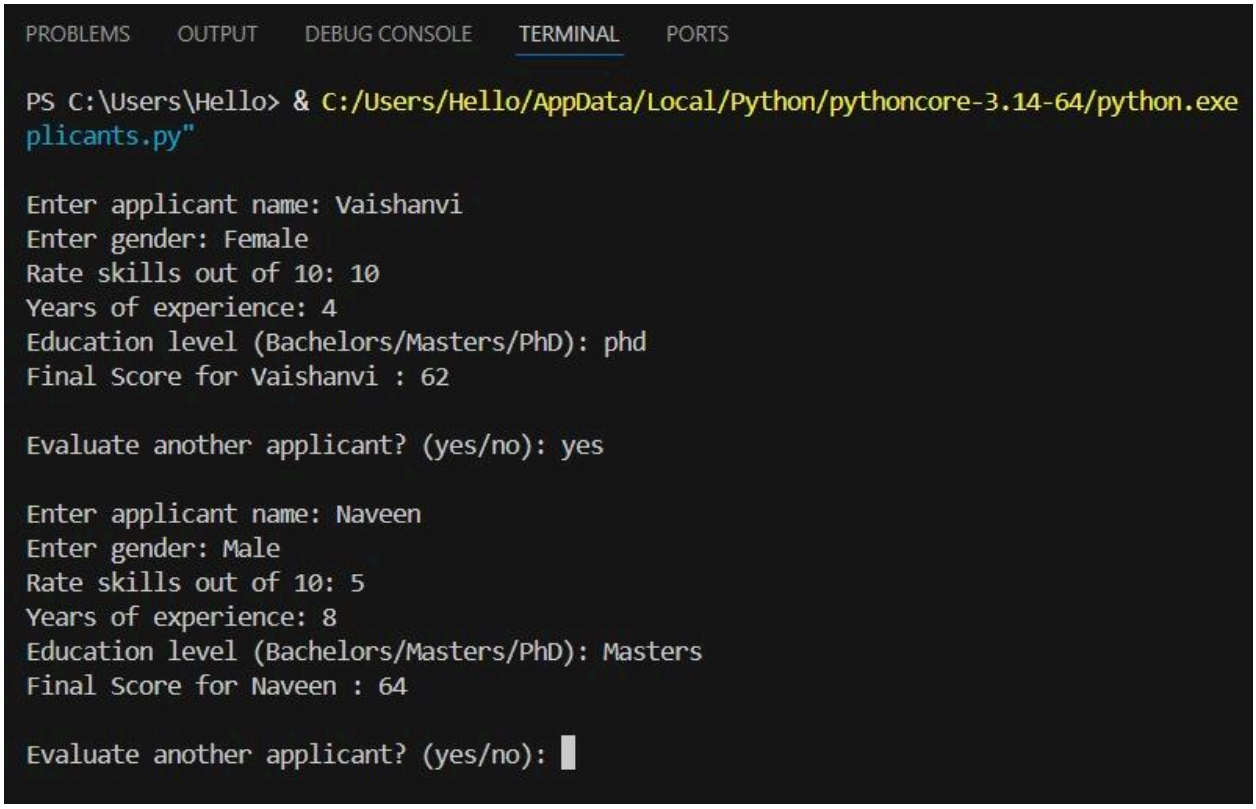
Prompts:
Write a Python console program that scores job applicants in a while True loop. Take inputs: name, gender, skills (out of 10), years of experience, and education (Bachelors/Masters/PhD). Initialize score = 0, add skills × 5 and experience × 3, add 10 for Masters and 15 for PhD, and add 5 extra points if gender is male (for bias analysis). Print the final score with the applicant name. After each applicant ask "Evaluate another applicant? (yes/no):" and stop if the answer is not "yes". Include comments.

Code:

C: > Users > Hello > # Loop to process multiple applicants.py > ...

```python
# Loop to process multiple applicants
while True:
    print("\n=== Applicant Evaluation System ===")
    # Taking applicant details
    name = input("\nEnter applicant name: ")
    gender = input("Enter gender: ")
    skills = int(input("Rate skills out of 10: "))
    experience = int(input("Years of experience: "))
    education = input("Education level (Bachelors/Masters/PhD): ")

    # Initial score
    score = 0

    # Score calculation based on skills and experience
    score += skills * 5
    score += experience * 3

    # Adding points based on education
    if education == "Masters":
        score += 10
    elif education == "PhD":
        score += 15

    # Biased logic (for analysis purpose)
    if gender.lower() == "male":
        score += 5

    # Display final score
    print("Final Score for", name, ":", score)

    # Ask to continue
    again = input("\nEvaluate another applicant? (yes/no): ")
    if again.lower() != "yes":
        print("System closed.")
        break
```

Output:



Justification:
This program scores job applicants using skills, experience, and education. A loop allows evaluating multiple applicants. The gender condition is intentionally added to demonstrate bias, helping analyze fairness and ethical issues in AI systems.

Task 5: Inclusiveness and Ethical Variable Design
Scenario
Inclusive coding practices avoid assumptions related to gender, identity, or roles and promote fairness in software design.
Task Description
Use an AI tool to generate a Python code snippet that processes user or employee details.
Analyze the code to identify:
• Gender-specific variables (e.g., male, female)
• Assumptions based on gender or identity

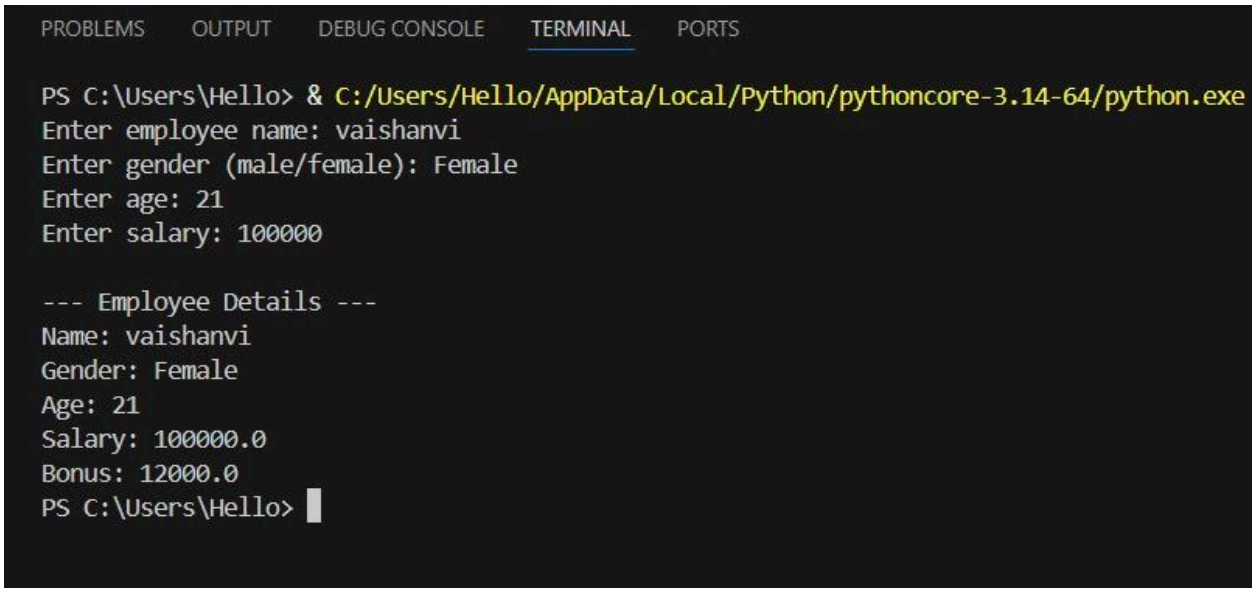
Prompt:
Write a simple Python program for an Employee Record System.The program should,Take employee name, gender (male/female), age, and salary as inputs,Use gender-based conditions to calculate bonus.Give 10% bonus for males, 12% bonus for females, and 8% for others,Display all employee details and the bonus,Keep the program basic using if-elif-else and input().

Code:

C: > Users > Hello > Employee record.py > ...

```python
# Employee Record System (Non-Inclusive Version)
name = input("Enter employee name: ")
gender = input("Enter gender (male/female): ")
age = int(input("Enter age: "))
salary = float(input("Enter salary: "))

# Gender-based bonus logic
if gender.lower() == "male":
    bonus = salary * 0.10
elif gender.lower() == "female":
    bonus = salary * 0.12
else:
    bonus = salary * 0.08

print("\n--- Employee Details ---")
print("Name:", name)
print("Gender:", gender)
print("Age:", age)
print("Salary:", salary)
print("Bonus:", bonus)
```

Output:



Justification:
The original program is non-inclusive because it uses gender-specific inputs and applies different bonus calculations based on gender. This assumes a gender binary and introduces potential bias in decision-making. Such logic is not fair or necessary for employee evaluation, as job-related factors like experience or performance should be used instead.